Remarks

Applicants respectfully request reconsideration of the present application in view of the foregoing amendments and the following remarks. Claims 1, 3-7, 9-10, 12, 13-15, 17-21, 24, 27-29, and 31-37 are pending in the application. Claims 1, 3-7, 9, 10, 12, 13, 15, 17-21, 24, and 26-37 are rejected. No claims have been allowed. Claims 1, 6, 12, 14, and 24 are independent. Claims 2, 8, 11, 13, 16, 22, 23, 25, 26, and 30 have been canceled. Claims 1, 6, 12, 14, and 24 have been amended.

Cited Art

The Office action dated July 14, 2009 (hereinafter "Action") applies the following cited art: U.S. Patent App. No. 2002/0169999 to Bhansali et al. (hereinafter "Bhansali"), U.S. Patent App. No. 2003/0101438 to Mishra et al. (hereinafter "Mishra"), U.S. Patent No. 6,560,774 to Gordon (hereinafter "Gordon"), Serge Lidin, *Inside Microsoft .NET IL Assembler* (hereinafter "Lidin"), and U.S. Patent No. 6,412,020 to Leach et al. (hereinafter "Leach").

The "IUnknown" described by *Leach* does not teach or suggest representing types in an intermediate language including an unknown type.

The Action asserts that the IUnknown interface function described by Leach is "a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known." (Action, Pgs. 3-4). Applicants disagree. The cited portions of *Leach* describe the IDatabase and IUnknown "interfaces." (*Leach*, Col. 6, lines 59-67). According to *Leach*, when a word processing program needs to determine whether the IDatabase interface is supported, another interface called IUnknown is defined "with a function member that indicates which interfaces are implemented for the object." (*Leach*, Col. 6, lines 59-67). The IUnknown interface returns a pointer or a false, depending on whether the interface is supported. (Col. 7, lines 9-16).

As will be discussed further below, the cited portions of *Leach* have nothing to do with determining whether an intermediate language representation having a known type should be associated with an unknown type. While the name of the interface happens to be called "IUnknown" the type of IUnknown is never changed—it remains an object interface. Furthermore, *Leach* has nothing to do with intermediate languages; instead, *Leach* describes

object-oriented programming techniques related to programming languages, not intermediate languages.

Patentability of Claims 1, 3-5, 12, 29, and 33-35 under 35 U.S.C. § 103(a) over *Bhansali* in view of *Mishra*

The Action rejects claim 1, 3-5, 12, 29, and 33-35 for allegedly being unpatentable under 35 U.S.C. § 103(a) over *Bhansali* in view of *Mishra*. (Action, Pgs. 4-7). The Action's rejections are respectfully traversed, for at least the following reasons:

Independent Claim 1 Is Allowable.

> A method of type-checking a code segment written in a programming language comprising:
>
> translating the code segment from the programming language to one or more representations of a typed intermediate language, wherein the one or more representations of the typed intermediate language are capable of representing programs written in a plurality of different source languages, and wherein the one or more representations comprise a first object having a known type, and wherein the translating comprises:
>
> determining that the first object will be type-checked as an unknown type, and
>
> based on the determining, designating the first object as having the unknown type by dropping the known type of the first object and replacing the known type with the unknown type; and
>
> type-checking the one or more representations based on a rule set, wherein:
>
> the rule set comprises rules for type-checking the type designated as the unknown type,
>
> the unknown type indicates that an element of the representation is of a type that is not known, and
>
> the first object is type-checked as having an unknown type, based on the designating.

The Action concedes that *Bhansali* does not teach or suggest "determining that the first object will be type-checked as an unknown type" and "based on the determining, designating the first object as having the unknown type." (*See* Action, Pgs. 5-6). *Bhansali*'s deficiencies are not cured by *Mishra*. According to *Mishra*, a "converter" is used to convert operations and/or code from bytecode or an object-oriented programming language ("OOPL") to intermediate language code. (*Mishra*, ¶¶ 0041-0045). What the cited portions of *Mishra* describe is the handling of

"arrays created through a reflection API." (*Mishra*, ¶ 0059). This includes handling of cases where a class "name is not known until runtime," a "field name is unknown to your program until runtime," "method is not known until runtime," and "create a new array, whose size and component type are not known until runtime." (*Mishra*, ¶ 0059). Hence, what *Mishra* describes is how to handle converting object-oriented code (from bytecode or an OOPL) to an intermediate language when the properties of a given object are not known at compile time. *Mishra* states that this is done so that "[s]trict type checking at runtime for array element assignments in .NET™ framework ensures that an exception (*e.g.*, ArrayTypeMismatchException) gets thrown at runtime." (*Mishra*, ¶ 0052).

Thus, according to *Mishra*, objects for which type information is not known can be ignored by the converter until the type information is known, and then "[s]trict type checking" can be applied at runtime, after the type information is known. In contrast to the cited portions of *Mishra*, amended claim 1 is directed towards objects having a known type, which during translation, are designated as having the unknown type, and then type-checking the representation. Hence, claim 1 recites translating "one or more representations [that] comprise a first object having a known type," "determining that the first object will be type-checked as an unknown type," "based on the determining, designating the first object as having the unknown type by dropping the known type of the first object and replacing the known type with the unknown type."

Because the combination of *Bhansali* and *Mishra* does not teach or suggest, whether considered alone or in any combination thereof, the method of amended claim 1, the Action has not sufficiently stated a case for obviousness. For at least this reason, the allowance of claim 1 is respectfully requested.

Independent Claim 12 Is Allowable.

The Action rejects independent claim 12 as allegedly being unpatentable under 35 U.S.C. § 103(a) over *Bhansali* in view of *Mishra*. (Action, Pgs. 6-7). Applicants disagree, but in the interest of expediting prosecution, have amended claim 12 to recite:

> A method of translating types associated with a plurality of programming languages to types of an intermediate language, the method comprising:
>
> replacing the types associated with the plurality of programming

languages with the types of the intermediate language, wherein:

at least one of the plurality of programming languages is non-type-safe,

the types of the intermediate language comprise plural programming language specific primitive types associated with the plurality of programming languages and a type designated as an unknown type,

the type designated as the unknown type has size information associated with it,

at least one of the plural programming language specific primitive types is replaced with the unknown type, and

the size information comprises size information of a machine representation of the type designated as the unknown type.

The Action concedes that *Bhansali* does not teach or suggest "at least one of the plural programming language specific primitive types is replaced with the unknown type" as recited by amended claim 12. (Action, Pg. 7). *Bhansali*'s deficiencies are not cured by *Mishra*. As discussed above regarding amended claim 1, *Mishra* describes converting bytecode or OOPL source code to intermediate language code when some information about an object is not known until runtime, and then performing strict type checking at runtime (*i.e.*, after the information is known). (*Mishra*, ¶¶ 0052, 0059). Thus, according to *Mishra*, when the types of a language are not known, any type-checking being performed would not occur until runtime, after the types of the objects are known. Thus, neither *Bhansali* nor *Mishra* describe replacing known types with unknown types.

In contrast, the method recited by amended claim 12 translates types associated with a programming language, and replaces at least one of these types with an unknown type. Hence, neither *Bhansali* nor *Mishra* teach or suggest a method of translating types, wherein "at least one of the plural programming language specific primitive types is replaced with the unknown type," as recited by amended claim 12.

Further, the cited portions of *Mishra* concern a converter for Java, Visual C#, and Visual Basic, which, as understood by the Applicants, are type-safe programming languages. Thus, *Mishra* does not teach or suggest replacing the types associated with the plurality of programming languages, wherein "at least one of the plurality of programming languages is non-type-safe," as recited by amended claim 12.

Dependent Claims 3-5, 29, and 33-35 Are Also Allowable.

Claims 3-5, 29, and 33-35 depend from one of independent claims 1 or 12, and should be allowable for at least the reasons recited above regarding those respective claims. Claims 3-5, 29, and 33-35 are also allowable on account of the novel and non-obvious combinations of features recited by each respective claim. For at least these reasons, claims 3-5, 29, and 33-35 are in condition for allowance, and such action is respectfully requested.

Patentability of Claims 6, 7, 9, 10, 24, 26-28, 31, 36, and 37 under 35 U.S.C. § 103(a) over *Gordon* in View of *Lidin* in Further View of *Leach*

The Action rejects claim 6, 7, 9, 10, 24, 26-28, 31, 36, and 37 for allegedly being unpatentable under 35 U.S.C. § 103(a) over *Gordon* in view of *Lidin* in further view of *Leach.* (Action, Pgs. 9, 11). The Action's rejections are respectfully traversed, for at least the following reasons:

Independent Claim 6 Is Allowable.

The Action rejects independent claim 6 as allegedly being unpatentable under 35 U.S.C. § 103(a) over *Gordon* in view of *Lidin* in further view of *Leach.* (Action, Pgs. 9). Applicants disagree, but in the interest of expediting prosecution, have amended claim 6 to recite:

> A method of selectively retaining type information during compilation in a code segment written in a programming language, the method comprising:
>
> translating the code segment from the programming language to one or more intermediate language representations;
>
> for each intermediate language representation, determining whether to retain type information for one or more elements of the intermediate language representation, wherein the one or more intermediate language representations include one or more elements having a known type;
>
> based on the determination, associating at least one of the one or more elements of the intermediate language representation with a type, designated as an unknown type, indicating the element can be of any type; and
>
> type-checking the one or more intermediate language representations based on a rule set, wherein the rule set comprises rules for type-checking the type designated as the unknown type.

Gordon, *Lidin* and *Leach* do not teach or suggest, whether considered alone or in any

combination "for each intermediate language representation, determining whether to retain type information for one or more elements of the intermediate language representation, wherein the one or more intermediate language representations include one or more elements having a known type" or "based on the determination, associating at least one of the one or more elements of the intermediate language representation with a type, designated as an unknown type, indicating the element can be of any type" as recited by amended claim 6.

As conceded in the Action, neither *Gordon* nor *Lidin* teach or suggest "a type, designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known." (Action, Pg. 10). Thus, *Gordon* and *Lidin* do not teach or suggest "based on the determination, associating at least one of the one or more elements of the intermediate language representation with a type, designated as an unknown type, indicating the element can be of any type" as recited by amended claim 6. These deficiencies are not cured by *Leach*.

The cited portions of *Leach* describe the IDatabase and IUnknown "interfaces." (*Leach*, Col. 6, lines 59-67). According to *Leach*, when a word processing program needs to determine whether the IDatabase interface is supported, another interface called IUnknown is defined "with a function member that indicates which interfaces are implemented for the object." (*Leach*, Col. 6, lines 59-67). The IUnknown interface returns a pointer or a false value, depending on whether the interface is supported. (Col. 7, lines 9-16). In contrast to amended claim 6, however, *Leach* does not teach or suggest "for each intermediate language representation, determining whether to retain type information for one or more elements of the intermediate language representation, wherein the one or more intermediate language representations include one or more elements having a known type" or "based on the determination, associating at least one of the one or more elements of the intermediate language representation with a type, designated as an unknown type, indicating the element can be of any type," as recited by amended claim 6.

The cited portions of *Leach* have nothing to do with determining whether an intermediate language representation having a known type should be associated with an unknown type. While the name of the interface happens to be called "IUnknown" the type of IUnknown is never changed—it remains an object interface.

Moreover, *Leach* has nothing to do with intermediate languages. Instead, *Leach* concerns object-oriented techniques:

> In the following, an object will be described as an instance of a class as defined by the C++ programming language. One skilled in the art would appreciate that objects can be defined using other programming languages.

(*Leach*, Col. 4, line 65 through Col. 5, line 2). Thus, *Leach* describes object-oriented programming techniques related to programming languages, not intermediate languages.

Further, even if *Leach* did describe a "type designated as the unknown type" (and it does not), *Leach* cannot be combined with *Gordon* and *Lidin*, because it describes techniques related to programming languages, not an intermediate language representation, as recited by amended claim 6.

Because the combination of *Gordon*, *Lidin*, and *Leach* does not teach or suggest, whether considered alone or in any combination thereof, the method of amended claim 6, the Action has not sufficiently stated a case for obviousness. For at least this reason, the allowance of claim 6 is respectfully requested.

Independent Claim 24 Is Allowable.

The Action rejects independent claim 24 as allegedly being unpatentable under 35 U.S.C. § 103(a) over *Gordon* in view of *Lidin* in view of *Leach*. (Action, Pg. 11). Applicants disagree, but in the interest of expediting prosecution, have amended claim 24 to recite:

> A method of representing types in an intermediate language comprising:
>
> defining a plurality of types to be associated with elements of the intermediate language, wherein one of the plurality of types indicates that an element of the intermediate language is associated with a type designated as an unknown type;
>
> wherein the type indicating that an element of the intermediate language is associated with the type designated as the unknown type has a size associated with it, wherein the size represents size of a machine representation of the type designated as the unknown type;
>
> wherein an element of the intermediate language that was previously associated with a known type is associated with the type designated as the unknown type;
>
> wherein the rule set further comprises rules for dropping type information for one or more elements of the representation by changing the known type of the one or more elements to the type designated as an unknown type.

The Examiner admits that neither *Gordon* nor *Lidin* teach or suggest "a type designated

as an unknown type, wherein the unknown type indicates that an element of the representation if of a type that is not known." (Action, Pg. 12). Thus, *Gordon* and *Lidin* do not teach or suggest "wherein an element of the intermediate language that was previously associated with a known type is associated with the type designated as the unknown type." These deficiencies are not cured by *Leach*.

The cited portions of *Leach* describe the IDatabase and IUnknown "interfaces." (*Leach*, Col. 6, lines 59-67). According to *Leach*, when a word processing program needs to determine whether the IDatabase interface is supported, another interface called IUnknown is defined "with a function member that indicates which interfaces are implemented for the object." (*Leach*, Col. 6, lines 59-67). The IUnknown interface returns a pointer or a false, depending on whether the interface is supported. (Col. 7, lines 9-16). In contrast to amended claim 24, however, *Leach* does not teach or suggest "wherein an element of the intermediate language that was previously associated with a known type is associated with the type designated as the unknown type" as recited by amended claim 24.

The cited portions of *Leach* have nothing to do with modifying a known type to be an unknown type. While the name of the interface happens to be called "IUnknown" the type of IUnknown is never changed—it remains an object interface.

Moreover, *Leach* has nothing to do with intermediate languages. Instead, *Leach* concerns object-oriented techniques:

> In the following, an object will be described as an instance of a class as defined by the C++ programming language. One skilled in the art would appreciate that objects can be defined using other programming languages.

(*Leach*, Col. 4, line 65 through Col. 5, line 2). Thus, *Leach* describes object-oriented programming techniques related to programming languages, not intermediate languages.

Further, even if *Leach* did describe a "type designated as the unknown type" (and it does not), *Leach* cannot be combined with *Gordon* and *Lidin*, because it describes techniques related to programming languages, not intermediate languages.

Therefore, *Leach* does not teach or suggest "wherein an element of the intermediate language that was previously associated with a known type designated as the unknown type" as recited by amended claim 24.

Because the combination of *Gordon*, *Lidin*, and *Leach* does not teach or suggest, whether

considered alone or in any combination thereof, the method of amended claim 24, the Action has not sufficiently stated a case for obviousness. For at least this reason, the allowance of claim 24 is respectfully requested.

Patentability of Claims 14, 15, 17-21, and 32 under 35 U.S.C. § 103(a) over *Gordon* in View of *Lidin* in Further View of *Leach* in Further View of *Mishra*

The Action rejects claim 14, 15, 17-21, and 32 for allegedly being unpatentable under 35 U.S.C. § 103(a) over *Gordon* in view of *Lidin* in further view of *Leach* in further view of *Mishra.* (Action, Pgs. 9, 11). The Action's rejections are respectfully traversed, for at least the following reasons:

Independent Claim 14 Is Allowable.

The Action rejects independent claim 14 as allegedly being unpatentable under 35 U.S.C. § 103(a) over *Gordon* in view of *Lidin*, in further view of *Leach*, in further view of *Mishra* (Action, Pg. 16). Applicants disagree, but in the interest of expediting prosecution, have amended claim 14 to recite:

> A computer system for type-checking an intermediate representation of source code in a compiler comprising:
> a computer-readable storage medium containing one or more types associated with elements of the intermediate representation, wherein at least one of the types, designated as an unknown type, indicates an element can be of any type;
> a computer-readable storage medium containing one or more rule sets comprising rules associated with the type, designated as the unknown type, indicating an element can be of any type; and
> a type-checker module, wherein the type-checker is configured for applying the one or more rule sets to the elements of the intermediate representation, wherein the type-checker module selectively retains type information for some elements of the intermediate representation and selectively does not retain type information for at least one element of the intermediate representation by replacing a known type associated with the at least one element with the type, designated as the unknown type, indicating the at least one element can be of any type, wherein the type-checker is configured to apply the one or more rule sets after the replacing the known type with the unknown type.

The Examiner admits that *Gordon*, *Lidin*, and *Leach* do not teach or suggest the language of claim 14, which recites a type-checker module that selectively does not retain type

information for at least one element of the intermediate representation by replacing a known type associated with the at least one element with the type, designated as the unknown type, indicating the at least one element can be of any type. (Action, Pg. 16).

The deficiencies of *Gordon*, *Lidin*, and *Leach* are not cured by *Mishra*. As discussed in more detail above regarding amended claim 1, *Mishra* describes a "converter" that converts operations and/or code from bytecode or an object-oriented programming language ("OOPL") to intermediate language code. (*Mishra*, ¶¶ 0041-0045). *Mishra* states that the converter handles cases where a class "name is not known until runtime," a "field name is unknown to your program until runtime," "method is not known until runtime," and "create a new array, whose size and component type are not known until runtime." (*Mishra*, ¶ 0059).

Thus, *Mishra* describes a system where unknown object information can be handled by the converter until the type information is known, and then applying "[s]trict type checking at runtime" when the type information is known. In contrast, amended claim 14 is directed towards a computer system, including a type-checker module that "selectively does not retain type information for at least one element of the intermediate representation by replacing a known type associated with the at least one element with the type, designated as the unknown type, indicating the at least one element can be of any type, wherein the type-checker is configured to apply the one or more rule sets after the replacing the known type with the unknown type."

Because of the combination of *Gordon*, *Lidin*, *Leach*, and *Mishra* does not teach or suggest, whether considered alone or in any combination thereof, the method of amended claim 14, the Action has not sufficiently stated a case for obviousness. For at least these reasons, the allowance of claim 14 is respectfully requested.

Dependent Claims 15, 17-19, 20-21, and 32 Are Also Allowable.

Claims 15, 17-19, 20-21, and 32 depend from independent claim 14, and should be allowable for at least the reasons recited above regarding those respective claims. Claims 15, 17-19, 20-21, and 32 are also allowable on account of the novel and non-obvious combinations of features recited by each respective claim. For at least these reasons, claims 15, 17-19, 20-21, and 32 are in condition for allowance, and such action is respectfully requested.

Interview Request

If the claims are not found by the Examiner to be allowable, the Examiner is requested to call the undersigned attorney to set up an interview to discuss this application.

Conclusion

The claims in their present form should be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By / Mark W. Wilson /
Mark W. Wilson
Registration No. 63,126